Manual

Version 1.0.2

Simple Soccer Football Kit

Please contact me if you have any questions:
diorgo@gmail.com

Unity forum thread:
http://forum.unity3d.com/threads/simple-soccer-football-kit.402264/

Contents

Overview....4
How to Setup the kit....5
Run the Setup menu....5
Edit the Layer Collision Matrix....6
InputManager....6
Adding your own assets to the project....7
Editing the default assets....7
Upgrade Guidelines....8
Before you upgrade / starting a new project....8
Upgrading....8
Resources folder and Resource Manager....9
Resource IDs....10
Scene IDs....10
Spawn Persistent Prefabs....11
How to start a match....12
Debug Match Prefab (testing a field scene in the editor)....13
Match Settings....14
Match Input Manager....15
Match Camera....16
Different camera views....16
Tournaments....17
Pivot Points and Axes....18
World Units....20
Layers and Collision....21
Fields and Stadiums....22
Field Zones....23
Teams....24
Players....24
Team Formations....24
Players....26
Mesh....26
Animations....26
Skills....26
Player AI....27
Position Properties....29

Shadows.....30
Real Time Shadows.....30
Fake Shadows.....30
Menus.....31
Menu design notes.....31

Overview

This manual contains information about Simple Soccer Football Kit. The kit is a Unity asset, soccer starter kit, that makes it easy for you to create soccer games. It is primarily aimed at simple, action soccer games.

How to Setup the kit

Import the package into your Unity project.

The package contains 2 folders:

- SimSoc: The main folder.
- Gizmos: Contains a few images used by the Formations Editor.

Run the Setup menu

After importing the package, run the following menu to add the scenes to the build settings, and to setup the layers:

Tools\Simple Soccer\Setup Kit

It will add the following scenes to the build settings (Startup must be the first scene):

- SimSoc\Scenes\Startup
- SimSoc\Scenes\UI\LoadingScreen
- SimSoc\Scenes\UI\MainMenus
- SimSoc\Scenes\UI\SplashScreen
- SimSoc\Scenes\Fields\BlueField
- SimSoc\Scenes\Fields\RedField

It will also add the following layers:

- World
- Players
- Ball

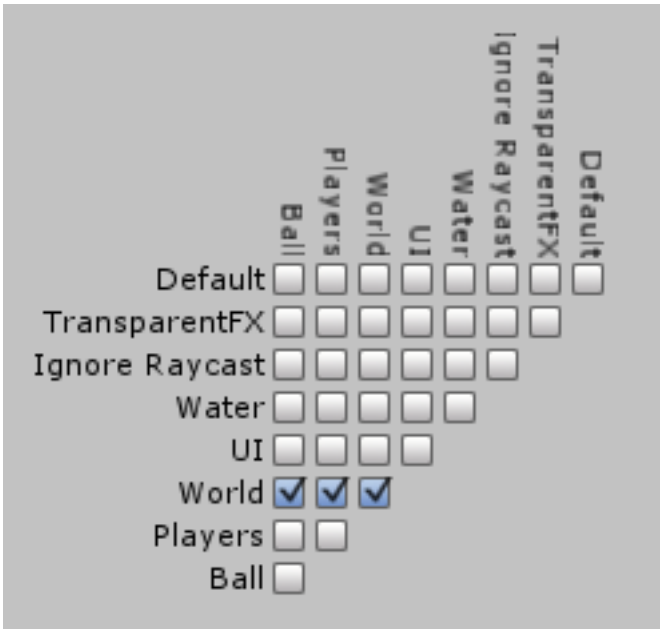
Edit the Layer Collision Matrix

Edit the layer collision matrix via the menu item:

Edit\Project Settings\Physics

The layers collision should be set up like this:

The World layer can collide with the Players, Ball and World layers.



See the **Layers and Collision** section below for more details.

InputManager

Importing into an empty project

After importing you will find the file "InputManager.zip" in the SimSoc folder. The zip contains the **InputManager.asset** file. Close the Unity project, then extract the **InputManager.asset** file and move it to your project's **ProjectSettings** folder.

InputManager.asset contains the input settings for the game (e.g. joystick buttons for 2 joysticks). These have been setup via Unity's Input Manager.

Importing into an existing project

If you have setup your own input settings in Unity's Input Manager then you should not use the **InputManager.asset** supplied with the kit. Instead you need to change the names of the buttons and axes on the **Match Input Manager** prefab.

For example: The kit uses "J1_Fire1" as default to read the first joystick's fire button. But you might have a joystick set up as "Joystick_Fire1". So you need to change "J1_Fire1" on the prefab to "Joystick_Fire1".

See the **Match Input Manager** section below for more info.

Adding your own assets to the project

Try to keep all your assets out of the SimSoc folder, to make it easier to upgrade the kit in the future. (See ***Upgrade Guidelines*** below.)

Editing the default assets

If you want to edit any of the default assets (e.g. teams, players, balls, global settings), rather try to make duplicates and edit the duplicates. Also keep the duplicates outside the SimSoc folder. This is to prevent your changes being overwritten when you upgrade. Remember to change all references to the new duplicate assets.

Upgrade Guidelines

Please follow these guidelines for upgrading the kit.

Before you upgrade / starting a new project

1. Keep all your Assets outside the SimSoc folder, so that the kit can safely be updated without the risk of overriding your Assets.
2. If you want to make script changes:
   - Ideally, avoid script changes where possible.
   - Instead, derive your own class from the script and override its virtual methods. If the method you want to override is not virtual, change it to virtual and please send me an email so I can do the same in the kit for future releases. The same for private variables which you change to protected. (If you change methods to virtual and variables to protected then you can email me a list of the changed ones and I'll integrate it into the kit.)
3. Make duplicates of the default prefabs and use the duplicates instead. Move the duplicates out of the SimSoc folder, and make sure you update the relevant references to use the duplicates.

Upgrading

1. Always make a backup of your project before upgrading the kit.
2. After upgrading it is a good idea to update the **Resource Manager** prefab via the menu: Tools\Simple Soccer\Scan Resources

Resources folder and Resource Manager

The following assets are stored in the Resources folder (or sub-folders):

- Teams prefabs
- Player prefabs
- Ball prefabs

The **Resource Manager** prefab is also stored in the Resources folder. The Resource Manager keeps a list of all the resources mentioned above. The game uses the Resource Manager to dynamically load resources.

The Resource Manager is automatically updated when resources are added or changed.
You can also manually update it via the menu:
Tools\Simple Soccer\Scan Resources

You can get info about a resource via the Resource Manager.

Example: If you have a team which has the ID “blue team” then you can use the following method to get the team's info (which includes the team's path and name):

SsResourceManager.Instance.GetTeam(“blue team”)

Or you could use the following to get info about a random team:

SsResourceManager.Instance.GetRandomTeam()

The main purpose of the Resource Manager is to avoid loading prefabs into memory until they are actually needed. It allows you to get info about a prefab before the prefab is loaded (e.g. you can get a list of all the team names to display on a menu).

Tip: If there are any default resources you are not going to use (e.g. teams), then delete them from of the Resources folder so they are not included in the builds. When you update the kit then un-tick them during the import so they are not imported again.

Resource IDs

Each resource has a unique ID.
Examples:

- Each team has an ID (e.g. “red team”, “blue team”).
- Each player has an ID (e.g. “red p1”, “red p2”, “red goalkeeper”, “bob”).
- Each ball has an ID (e.g. “white ball”, “green ball”, “metal ball”).

Examples where IDs are used:
The IDs are used by the menus to reference the resources. The game also uses the IDs when it selects a random resource (e.g. random ball). The tournament also saves/loads the team IDs in the matches. The Resource Manager also uses the IDs to store info about each resource.

Scene IDs

Each field scene also has a unique ID which is specified on the **Scene Manager** prefab.

Please remember to add new scenes to the **Scene Manager** prefab.

Spawn Persistent Prefabs

Each scene must contain the **Spawn Persistent Prefabs** prefab, which is located in the Prefabs folder. This prefab will spawn persistent game objects such as the **Global Settings**, so that they exist in all scenes, for the duration of the game.

By default, the only scene that does not contain the prefab is the **Startup** scene, to keep the scene as small as possible. However it is fine to add the prefab to the scene if you add other objects to the scene that need to reference any of the persistent prefabs (e.g. if you need to reference the global settings).

The Startup scene can be used to initialise plugins.

How to start a match

Call the **SsMatch.StartMatch** or the **SsMatch.StartMatchTest** method to start a match, from within any scene.

There is also a method **SsMatch.StartMatchFromMenus** which can be used, but it is dependent on a few settings from the menus (such as the selected teams).

Differences between the two methods:

SsMatch.StartMatch	**SsMatch.StartMatchTest**
Ideally used for the final game.	Ideally used for testing new fields, teams, players or balls.
It loads assets from the Resources folders. Therefore uses string file names as parameters.	It clones assets from prefabs. Therefore uses prefab references as parameters.
Takes longer to setup, because you need to put the assets in the Resources folder and use their file names as parameters to StartMatch.	Easier to setup and test, simply pass prefabs as parameters to StartMatchTest. Therefore it is better suited for quick testing.
Only loads assets as needed, therefore uses less memory.	Potentially uses more memory. Unity loads an asset prefab as soon as there is a reference to the prefab (whether you use the prefab or not). Example: If you use prefabs in your menus to select teams then Unity loads all the teams. So when you call StartMatchTest with the 2 selected teams, the other teams are still in memory. However, they will be unloaded when a new scene loads. So this method is fine if you have only a few teams and players.

Note: Make sure you set the relevant match settings before you start a match. Some

of the settings can be shown on a menu for the user to change. See the ***Match Settings*** section.

Debug Match Prefab (testing a field scene in the editor)

To make it easier to test a field/stadium scene, you can drag the **Debug Match** prefab into the scene. The prefab will start a new match when you run the scene in the editor.

On the prefab you specify which 2 teams you want to use in the match, and the control type for each team (e.g. human controlled keyboard/mouse, or AI controlled). You also specify which ball to use.

The **Debug Match** prefab uses the **EditorOnly** tag, so it will not be included in the scene when you do a build. Although, it is safer to delete the object from the scene when you are finished testing the scene.

Debug Output

The prefab has options to display debug output to the screen and in the scene. It may be useful when you are setting up the field, or tweaking and testing player AI settings.

Match Settings

The match settings are defined on the **Match Settings** prefab. The prefab is loaded at the start of the game (via the **Global Settings** prefab) and remains in memory for the duration of the game, therefore can be accessed at any time.

The match settings can be changed before a match starts. For example, you may show a menu that allows the user to change some of the settings (e.g. the duration of the match).

The tournament settings are also on this prefab.

The match settings can be accessed via the variable:
SsMatchSettings.Instance

Match Input Manager

The **Match Input Manager** prefab (script **SsMatchInputManager**) contains methods for reading input from the various hardware (e.g. keyboard, mouse, gamepad, touch screen, accelerometer).

You can override the class and methods for more specific input devices, or to change which named buttons/axes are used. (The named buttons/axes are setup in Unity's Input Manager via the menu: Edit\Project Settings\Input)

If you create a duplicate/new prefab, remember to change the reference to it in the **Match Prefabs** prefab.

Match Camera

The match camera prefabs are located in the Prefabs folder. There are a few different prefabs, including top down cameras. The default prefab is the **Match Camera** one.

The camera contains various properties. See the **SsMatchCamera** component for details.

Different camera views

The camera can be positioned on any side of the field, changing the direction of the match.
Example: Default goal posts are on the left and right side, but they can be set to be at the top and bottom. This is set via the **Play Direction** and **Follow Offset** properties on the camera.

The various camera prefabs are for the combinations of: side view, top down view and the different sides of the field.

Technical notes:
In the source code, "left team" refers to the team playing from -X to +X, while "right team" refers to the team playing from +X to -X. This is based on the very first camera view included with the kit, with the camera positioned on the side of the field looking down the +Z. So -X was on the left side and +X was on the right side.

The left and right teams always refer to their play direction on the X axis, no matter from which side the camera is looking.

Tournaments

There are 2 types of tournaments:

- Log based tournaments (e.g. leagues)
- Single-Elimination tournaments

You can create many tournaments based on the two types. Each tournament has its own settings.
Example: You can have a short log tournament which has 4 teams, and a longer log tournament which has 8 teams.

Log based tournaments
The teams are put into a log. Points are awarded for winning and drawing matches. The 2 teams with the most points will play in the final. You can specify how many times you want teams to face each other (e.g. how many times must Team A face Team B in the tournament). The more times teams face each other, the more matches there will be.

Single-Elimination tournaments
Teams play against each other and the losing teams are eliminated from the tournament. The winning teams move onto the next round, until only 2 teams remain to play in the final.

The tournament settings are on the **Match Settings** prefab.

Pivot Points and Axes

The player's pivot must be on the ground, in the middle of his feet.

The ball's pivot must be in its centre.

The player's axes must be aligned so that Z points forward, X to the right and Y up. If your mesh's axes are not aligned this way, then there is an option to spawn and attach the mesh as a child object to the player (which is the preferred method). See the **Mesh Child Or Prefab** property on the **SsPlayer** component.

Alternatively, you can manually attach the mesh as a child to the player and rotate (and position) it as needed.

The ball's axes must also be aligned so that Z points forward, X to the right and Y up.

World Units

The game works with real world units:
1 unit in Unity = 1 metre

Therefore:

- An average size human will be about 1.7 metres tall.
- An average size field will be about 100 metres long, and 65 metres wide.

And standard gravity is 9.81 metres per second.

If you change the units (e.g. everything is larger) then make sure you correctly adjust the relevant properties (e.g. player speeds, gravity, etc.).

Please note that physics might behave strange when not using the default world scale.

Layers and Collision

The kit uses these layers during a match:

- World: For scene objects, such as the ground.
- Player: For the players.
- Ball: For the ball.

The game does not use physics to detect collision between the players and ball, and between the players and players. This is mainly for optimisation.

By default, physics collision between the players layers are turned off, because the game is more fun when players do not block each other's movement. However you can turn it on if you want a more realistic game, via the menu:
Edit\Project Settings\Physics

Fields and Stadiums

A match is played on a field (or a stadium). Each field has its own scene.

The scene must contain at least the following game objects:

Field Properties
A game object that has a **SsFieldProperties** component. The component has various properties that define the field, such as the play area, position of the centre mark, positions of the penalty spots, positions of the goal posts, etc.

See the SSFieldProperties component for more details about each property.

You can use the following menu to add a Field Properties game object to the scene, which will contain some default settings (which you can tweak):
Tools\Simple Soccer\Add Field Properties

Field mesh(es)
The actual field mesh (or meshes), which include the ground, goal posts, stands, flood lights, etc.

Colliders
The colliders for the ground, goal posts and any objects that limit the ball and players' movement.

The colliders may be attached to the relevant field meshes.

Ideally, the colliders should be simple colliders (e.g. BoxColliders) to speed up physics.

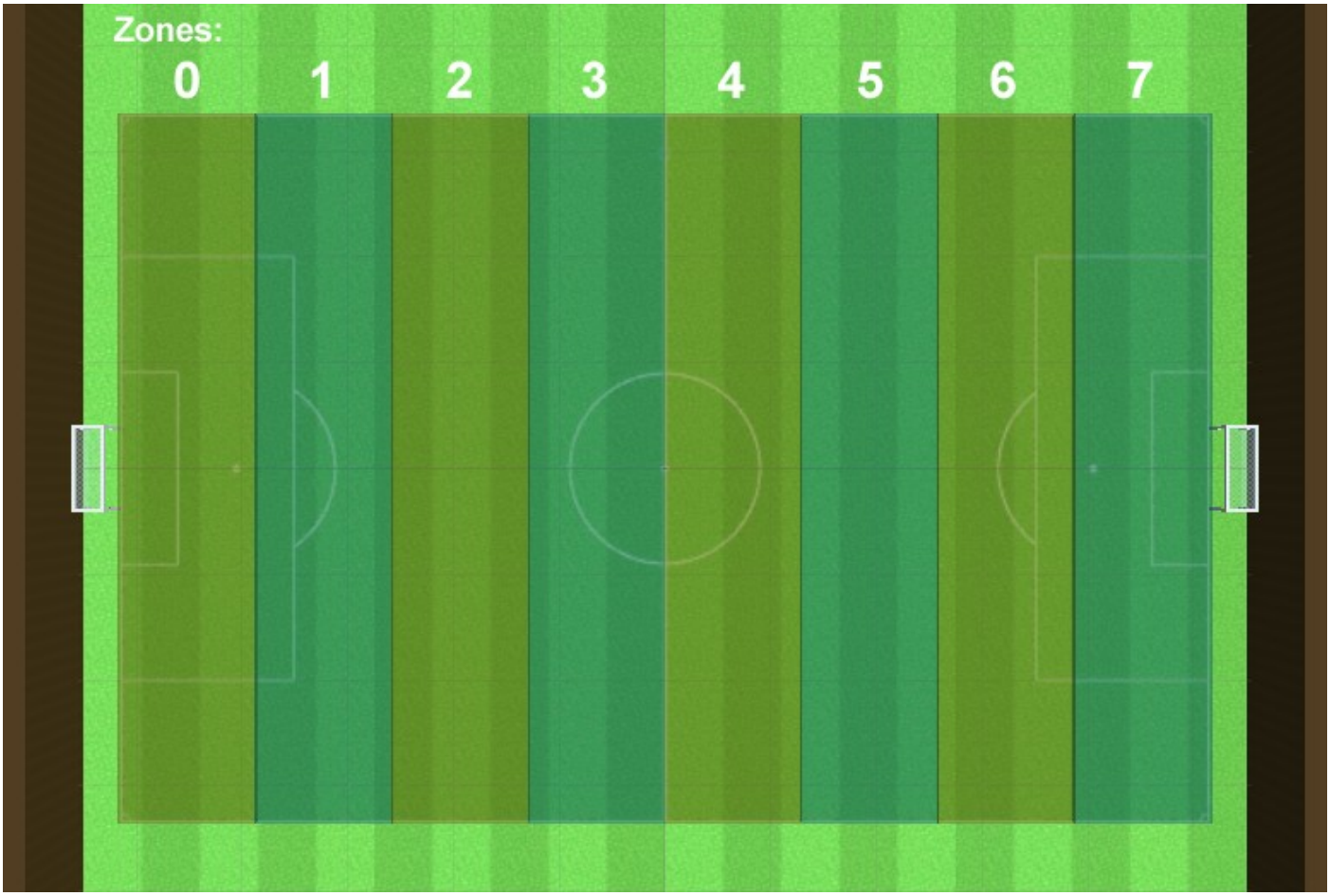
All the colliders should use the “World” layer.

Field Zones

Internally, the field is divided up into 8 vertical zones. Each half of the field has 4 zones.
Zone 0 is at the team's goal posts and zone 7 is at the other team's goal posts.

The player skills has some zone dependant properties.

Examples of when zones are used:
Zones are used by AI (e.g. to detect where ball is, when to shoot) and for positioning players.



Teams

The default teams and players prefabs are located in the folder:
Assets\SimSoc\Resources\Teams

Players

A team must have at least 2 player prefabs: a goalkeeper and a non-goalkeeper. When a match starts then it clones the prefabs to fill up the required number of players.
Example: If a match needs 11 players then the goalkeeper will be cloned once and the non-goalkeepers will be cloned to fill the other 10 spots (as is required by the formation).

Team Formations

Team formations are setup on the **Formation Manager** prefab. You can edit existing formations and add new ones.

You can use the scene **"Formation Editor"** to edit the formations. This scene is only used for editing the formations and should not be included in the build settings.

Each formation has a unique ID which the game uses to reference the formation.

Tip: If you run the **"Formation Editor"** scene then it will display invalid or duplicate IDs in the console.

You can let the user select formations via a menu.

The formations can also be changed any time during the match by calling the team's **SetFormation** method. Ideally they should only be changed at kickoff (e.g. at half time or after a goal), because some players may be disabled if they are not required for the new formation, or new ones added if the formation requires them. (Example: the current formation might have 3 forward players, but the new formation only has 2

forward players.)
So you want to avoid players disappearing/appearing in the middle of a match, but rather have them disappear/appear when the camera changes.

Players

Mesh

The player's mesh (e.g. FBX) must be attached as a child object to the player game object, or it can be dynamically spawned and attached (by setting **Mesh Child Or Prefab** on the player component).

The main reason it is a child is so that the mesh can be rotated independently to the player's game object (e.g. goalkeeper rotates when diving).

Animations

The player has various states that can be linked to animation names. This allows you to specify which animation must be played for each state.
Example states: idle, run, kick, slide tackle, throw in.

Use the **SsPlayerAnimations** component (attached to the player) to link states with animations. If no link is specified for a specific state, then the game tries to use the state as the animation name (e.g. idle state will try to play the "idle" animation).

In most cases, when the player does not have an animation for a specific state then the idle state's animation will be played.

Skills

The player's skills are set on the **SsPlayerSkills** component.

Examples of properties on the skills component:

- run speed
- slide tackle speed
- slide tackle distance

- can the player change direction while sliding
- when an AI should shoot to goal
- can the AI pass to the foremost player on the field

A player can have more than 1 skills component, but only 1 will be used during the match. The component has a **When To Use Skill** property which specifies when the component must be used.

Example: A player might have 4 components attached:

- 1 which will be used for a team controlled by a human.
- 1 which will be used for a team controlled by the computer, for Easy game difficulty.
- 1 which will be used for a team controlled by the computer, for Medium game difficulty.
- 1 which will be used for a team controlled by the computer, for Difficult game difficulty.

The skills component can be attached to the player game object (or one of its children), or it can be attached to the team game object (or one of its children). If it is attached to the team then all the players who do not have skills components will use the ones attached to the team. This is an easy way to give all the players in the team the same skills.

Player AI

AI Classes

The following AI classes are available:

SsPlayerAi
The base AI class used by all the other classes.

SsPlayerAiForward
Class for forward players.

SsPlayerAiMidfielder
Class for midfielder players.

SsPlayerAiDefender
Class for defender players.

SsPlayerAiGoalkeeper
Class for goalkeepers.

Custom AI
If you want to create your own AI behaviour then you can derive your own AI class from one of the existing classes and attach it to the player.

Position Properties

The player has an array of position properties, which specify which positions the player can play.
Examples:

- If the player can only play in the forward position then only add forward to the array.
- If the player can play forward or midfield then add a forward and a midfield to the array.

It also specifies the AI component to use for each position (e.g. forward position can use the SsPlayerAiForward component, defender position can use the SsPlayerAiDefender component or a custom defender component, etc.).

If no AI component is attached to the player then the game will attach one of the following, based on the player's position:
SsPlayerAiForward
SsPlayerAiMidfielder
SsPlayerAiDefender
SsPlayerAiGoalkeeper

Shadows

Real Time Shadows

Like any Unity game, you can use real time shadows in the game. But please note that shadows can be very slow on a mobile device.

If you use shadows and the game runs slow on mobile, try turning off the shadows (via a property on the **Global Settings** prefab) to test if it is the shadows that are slowing down the game.

Fake Shadows

The kit also supports fake shadows. The default ones are each a simple quad with a texture. The fake shadows can be set for the teams or players, and for the balls. If a team has a fake shadow then all the players who do not have fake shadows will use the team's.
The fake shadow prefabs are in the following folder:
Assets\SimSoc\Prefabs\Shadows

Menus

The kit has some basic menus which were created with Unity's native UI system. You can modify the menus* or replace the menus with whatever UI system you prefer.

**Rather make copies of the menus and modify the copies, so your changes are not lost when you upgrade the kit.*

Menu design notes

Menus have been designed at a resolution of **1920 x 1080**.

They have been designed to work on iPad aspect ration (1024 x 768) which is **1440 x 1080** (on a 1920 x 1080 screen). Therefore the max width of horizontally centred content is **1440**.
iPad was selected, because it has one of the narrowest aspect ratios available.

Tip: It is always a good idea to test your menus at iPad aspect ratio if you want to support multiple screen resolutions.

The minimum button size is **150 x 150** (at least the touchable area). Which automatically scales down to the minimum recommended button size on most mobile devices (e.g. it scales down to 88 x 88 pixels on iPhone 4). This does mean large buttons on desktop, but reduces the need to design two versions of each menu (one for desktop and one for mobile) or special code to scale the menus for mobile.
If your game is desktop only then you can make the buttons smaller.